IPv6 code checker tool

Salvatore Monforte (INFN CT)

EGEE 6, 25-29 September 2006, Geneva

www.eu-egee.org

Information Society and Media

Outline

- IPv6 reccomendations for developing "IPv6 ready" applications
- gLite WMS code "porting" impact
- glite IPv6 code checking tool

Moving to IPv6

- **Changing the network address data structure has a major impact on all aspects of IP inter-networking**
  - developer point of view
    - as first stage of transition we can start following few simple recommendations in order to
      - move to *"IPv6-ready" application while still running them on IPv4-based network*
        - such "IPv6 ready" applications can function both in IPv4 and IPv6 environments
        - migration to a pure IPv6 network without any modification to the application

Moving to IPv6

- **Impact of the UnixWare IPv6 implementation involves the following issues:**
  - IPv6 data structures and functions
    - new data structures required to hold the larger IPv6 address
      - ***in6_addr***, ***sockaddr_in6***
    - new and modified network API functions
      - ***IN6_IS_ADDR_V4MAPPED***, ***gethostbyname2***, ***getaddrinfo***, ***getnameinfo***, *etc etc*
  - Address and protocol families
    - new address and protocol family constants
      - ***AF_INET6***, ***PF_INET6***
  - Intercommunication between applications
    - within a mixture of IPv4 and IPv6 applications running on the same host passing open sockets is more complicated
      - ***IPV6_ADDRFORM***

Migration path to IPv6

- **One of the main objectives when implementing IPv6 in UnixWare was to provide a migration path to IPv6 while still enabling IPv4 applications to work**
  - this had the direct consequence of reducing the amount of effort required to “port” and existing IPv4 application to an “IPv6 ready” one
- **The following recommendations highlights the key aspects of porting IPv4 applications to IPv6**
  - Make sure you are using the correct data structures
  - Check for use of **INADDR_ANY** and **INADDR_LOOPBACK** for source and loop-back address selection
  - Modify any occurrence of IPv4 address and protocol family constants
  - Substitute the newer IPv6 functions for older IPv4 ones where necessary
  - Consider the use of new, more flexible functions which work in both IPv4 and IPv6 environments

Data structures

- **IPv4 applications use the** sockaddr_in **and** in_addr **structures to pass network address information between certain networking related functions**
- **IPv6 uses a larger address space and therefore uses different data structures**
  - ***in6_addr***
    - is used to store the 128−bit network address
  - ***sockaddr_in6***
    - is used to store the remaining details, previously stored by **sockaddr_in**, that is, length of the data structure, address family, flowinfo, port number and an **in6_addr** data structure.
- **Replace any occurrence of**
  - **sockaddr_in** and **in_addr** with the **sockaddr_in6 and in6_addr** structures

INADDR_ANY and INADDR_LOOPBACK

- **Any occurrence of** INADDR_ANY **or** INADDR_LOOPBACK **must be modified to use the newer global variables**
  - **in6addr_any** or
  - **in6addr_loopback** for assignments
- **If you need to initialize an** in6_addr **structure use**
  - either the **IN6ADDR_ANY_INIT**
  - or **IN6ADDR_LOOPBACK_INIT** macros

Address and protocol family constants

- ***AF_INET6***
  - IPv6 address family
- ***PF_INET6***
  - IPv6 protocol family
- **Replace in your application all occurrences of**
  - **AF_INET** with **AF_INET6** and
  - **PF_INET** with **PF_INET6**

Substituting old IPv4 functions with newer IPv6 versions

- **Three IPv4 functions have been succeeded by new functions:**
  - **gethostbyname** should be replaced with **gethostbyname2**
    - retrieves the network host entry referenced by a host name and its address family, which will be AF_INET6.
  - **inet_addr** should be replaced with **inet_pton**
    - interprets a character string representing an address and returns a value suitable for use as an internet address for both IPv4 and IPv6 address notations.
  - **inet_ntoa** should be replaced with **inet_ntop**
    - interprets an internet address and converts it to a character string for both IPv4 and IPv6 addresses.

- **You must use** inet_pton **and** inet_ntop **in your IPv6 application because the functions they replace (**inet_addr **and** inet_ntoa**) are not IPv6 aware.**

IPv6 and gLite WMS

- **gLite WMS is not a “simple” monolithic application**
  - is a mixture of
    - “proprietary” services (i.e. developed within EGEE)
    - third-party services
  - running together and interacting each other
    - fulfill user requests
    - supply end user with functionalities for authenticating, submitting jobs, inquiring job status etc, etc

IPv6 and gLite WMS

- **To better understand what is the "size" of the *problem***
  - consider the set of components needed to perform a complete build of the **glite-workload-manager** service
  - Taking into account the recommendations for the IPv4 to IPv6 porting
    - highlighting the possible *point of failure*
      - *for each component involved in the build, check the occurrence of "suspicious" IPv4 data-structure and functions…*

IPv6 and gLite WMS

	INADDR_	addr_in	F_INET	gethostbyname	inet_addr	inet_ntoa	
org.glite.wms-utils.tls	1	22	4	1	0	1	29
org.glite.wms-utils.jobid	0	0	0	1	0	0	1
org.gridsite.core	1	9	10	1	0	10	31
org.glite.security.voms	1	24	7	2	0	0	34
org.glite.security.gatekeeper	2	10	6	2	0	6	26
org.glite.security.gsoap-plugin	8	34	36	35	18	0	131
org.glite.jp.index	1	1	2	0	0	0	4
org.glite.jp.primary	1	1	2	0	0	0	4
org.glite.lb.server-bones	2	3	6	0	1	0	12
org.glite.lb.client	2	4	2	4	1	3	16
org.glite.lb.server	7	30	24	16	12	6	95
org.glite.lb.logger	1	2	2	0	0	0	5
org.glite.ce.blahp	8	15	20	1	0	1	45
org.glite.ce.monitor-client-api-c	0	0	1	1	0	0	2
org.glite.ce.cream-cli	0	0	0	1	0	0	1
org.glite.data.srm-cli	0	0	0	0	0	1	1
org.glite.data.io-protocol-rfio	0	0	0	1	0	0	1
org.glite.rgma.api-cpp	0	1	2	1	0	0	4
org.glite.rgma.api-c	0	1	3	2	0	0	6
org.glite.wms.thirdparty-globus_gridftp_server	3	35	20	8	10	13	89
org.glite.wms.thirdparty-bypass	1	8	7	3	0	0	19
org.glite.wms.ice	0	0	0	5	0	0	5
org.glite.wms.helper	0	0	0	1	0	0	1
org.glite.wms.manager-ns-commands	0	0	0	1	0	0	1
org.glite.wms.manager-ns-client	0	0	0	2	0	1	3
org.glite.wms.wmproxy	0	0	0	2	0	0	2
org.glite.wms.wmproxy-api-cpp	0	0	0	2	0	0	2
org.glite.wms.client	0	0	0	5	0	0	5
org.glite.gpbox.gsilib	1	25	3	1	0	0	30
	40	225	157	99	42	42	605

IPv6 and gLite WMS

- **making gLite WM an “IPv6 ready” system is not an *immediate* task**
  - identified dependencies within EGEE components can be easily “fixed” by modifying the relevant code
    - easier when the code is developed within JRA1
  - for third-party components the problem is a little bit tricky
    - since sources are not available IPv4 dependencies cannot be explicitly identified

```
glite_branch_3_1_0/org.glite.wms.ism >
nm ../../repository/globus/2.4.3-VDT-
1.2.2/rhel30_gcc32/lib/libldap_gcc32pthr.so.2 | grep gethostbyname
         U gethostbyname_r@@GLIBC_2.1.2
00019d7b T ldap_pvt_gethostbyname_a
```

The code checker tool

- **In order to perform an unattended IPv6 compliance check a dedicated code checker tools has been developed**
  - search “suspicious” IPv4 code patterns and function calls inside the source code
    - C/C++,Java,Python,Perl
  - It is just a simple bash script which should be executed in the main folder of the code to check
    - Considering the WMS build system directory structure
      - *at the same level of org.glite*

The code checker tool

- **Usage of the checker is straightforward:**
  - Copy the script in the main folder of the glite wms build system
    - same level of org.glite
  - Prepare a file containing the list of component to check IPv6 compliance for
    - To perform the check on org.glite.security components
      - ***find -type d -maxdepth 1 -name "org.glite.security*" | awk -F/ '{print$2}' > components***
  - **Issue the command**
    - **$./ipv6check.sh components**

The code checker tool

```
org.glite.security.voms
INADDR_                                                    [FAILED]
addr_in                                                    [FAILED]
F_INET$                                                    [PASSED]
gethostbyname                                              [FAILED]
inet_addr                                                  [PASSED]
inet_ntoa                                                  [PASSED]
Inet4Address                                               [PASSED]
inet_aton                                                  [PASSED]
gethostbyname_ex                                           [PASSED]
INADDR_BROADCAST                                           [PASSED]
0.0.0.0                                                    [FAILED]
127.0.0.1                                                  [PASSED]
255.255.255.255                                            [PASSED]
```

The code checker tool

- **Details on how to integrate the IPv6 checking in the current glite build system should be discussed**
  - define a new ant target for the current build system
    - "IPv6check"
      - *to be execute as next to "compile" target*
      - *since several code is autogenerated during the build it is available at compile completion time*
  - Include this check also in ETICS® ?